# Web Services, Business Objects and Component Models

## White Paper - July 2001

*By Philippe Mougin & Christophe Barriolade, Orchestra Networks*

Web services technologies such as SOAP, UDDI and WSDL will facilitate inter-enterprise cooperation on the Internet. Using Web services, your information system will be able to communicate much more easily with your partners' information system than in the past. This leads to an important question: what technology should be used for implementing Web services themselves? The general trend is to use top-level approaches which combine workflow and object modeling. This combination of Web services and a dynamic object approach is shaking up standard business component models i.e. EJB, COM and CORBA. New, more suitable technologies are emerging. Here, at Orchestra Networks, we use Web services to develop our technology. This document is a technical analysis based on our experience in this area.

www.**orchestranetworks**.com

*Foreword by*
*Alain Lefebvre, Groupe SQLI and*
*Jean-Christophe Cimetiere, CEO, TechMetrix Research.*

# Table of contents



 

# Foreword: The true nature of Web Services

Establishing a Web presence or developing distribution channels is both long and costly. On achieving these objectives you then must offer enough content and features to encourage visitors/customers/partners to venture beyond the homepage and to come back later.

Most Web sites which attract a lot of traffic use content syndication to enrich their own offering. They base themselves on Web services to extend their functionalities. Look at Yahoo! – most of its features come from other Web sites: travel, weather, paydirect, maps and Web searches using Google services. These features are integrated into the portal as native services. Each supplier therefore avails of greater exposure and/or income and enables Yahoo! to offer more services to users.

The example of Yahoo! illustrates the basic concept of Web Services. The problem is that, up until now, integrating services from different companies was very difficult. While sophisticated middlewares (EAI solutions, for example) are very useful for one-to-one integration, many-to-many integration is a whole different ballgame which requires specific development for each new service to be integrated. Similarly, for service providers, adding a new partner generally means customizing the source code.

The good news is that Web services are no longer a mere concept but a technological reality. The first chapters of this story have already been written by some software vendors (Microsoft, IBM-Apache, Ariba, Bowstreet...). All of these vendors are working to produce key standard protocols (i.e. SOAP, WSDL and UDDI) so as to make Web Services easier to use, and they also provide tools and solutions for creating Web Services.

## Web Services, a definition

According to Mark Colan (IBM), Web Services are "Internet-based modular applications that perform a specific business task and conform to a specific technical format." So, if certain processes from your applications can be invoked over the Internet, within a method and with a standard format, then you are already a ***server*** of Web Services. Similarly, if you call on certain processes external to your applications via Internet, then you are already a ***client*** of Web Services.

Everyone understands that for this sort of openness to work, you need standardized formats and methods. The good news is that this all exists and has been available since April of last year: SOAP 1.1.

SOAP is a Remote Procedure Call (RPC) protocol that uses standard Internet protocols for transport - either HTTP for synchronous calls or SMTP for asynchronous calls. SOAP uses XML for the envelope (i.e. the format of data transmitted). The abbreviation SOAP stands for Simple Object Access Protocol, but we could also translate it as "Service-Oriented Architecture Protocol."

 

Of course, this is all very well, but SOAP is still just an RPC, calling low-level functions and leaving pretty much everything up to developers. In order to make it easier to use, there is a format for describing services that can be invoked by SOAP — WSDL (see [1]).

WSDL can be seen as a complement to SOAP, as it facilitates interoperability between Web services. Like IDL (Interface Definition Language), which acts as a service describer with CORBA, WSDL (Web Service Description Language) is an XML syntax to describe Web services. The specifications for WSDL come from a joint initiative by Microsoft, IBM and Ariba. More and more SOAP implementations support this description language; with WSDL, applications that use SOAP can self-configure exchanges between Web services, while hiding most of the low-level technical details.

# Industry recognition and support

Not only does SOAP rely exclusively on well-established standards, but it is also widely recognized by the major market players such as Microsoft, IBM, Oracle, and even Sun. Even Microsoft's MS.net initiative has SOAP at its core, though the protocol has been taken on as a W3C project and has a multitude of open source implementations.

Within this context, SOAP and WSDL can be seen as the main pieces in the puzzle, but they do not form the entire schema on their own. As was the case for sites aimed at end-customers, Web Services dedicated to enterprises are set to experience a massive boom, if we are to believe the predictions. Industry professionals, confronted by an extremely rich offering in which it is hard to identify anything, will then have to face the eternal question of "Where — and how — does one find information?"

What the market needs is a simple service enabling players to:

- Find the right partner (the one offering the Web Service(s) you need)
- Get the information necessary to be able to embark on electronic exchange with the partner (access a WSDL document for instance).

This is where UDDI (Universal Description, Discovery, and Integration) comes in. UDDI was the brainchild of several of the giants of the IT industry (IBM, Microsoft and Ariba, the very same three that were behind WSDL... and it's no coincidence); it is essentially a vast worldwide database of services offered on the Web. At least 130 companies have promised to support UDDI. 36 were members of the original project and a further 94 have joined the movement since. The project has already managed to gather together companies such as Accenture, Cargill, Clarus, CommerceOne, Compaq, Dell, I2, ICG, Rational, Sabre, SAP, Sun, Tibco, WebMethods. All the players involved see it as a major step forward.
UDDI offers a technical framework that is independent from platforms and totally open, so that enterprises can find one another, define how they will interact on Internet, and define how information should be shared using a worldwide registration system. The result of this project, according to its promoters, will be that enterprises will be able to enter the B2B world a lot quicker.

 

So the technical schema that we can draw up for Web Services features the following:

- XML: format for data exchange and description
- SOAP: protocol for calling Web Services
- WSDL: format for describing Web Services
- UDDI: central organization for registering, finding and using Web Services

Thanks to SOAP, we finally have the middleware that will enable us to achieve client-server between applications and via Internet - and that is a major innovation! For at least fifteen years, we have been waiting for a simple, lightweight RPC that lets you outside of the local network. Previous attempts failed because they weren't simple, and they weren't standard.

It's not because CORBA, DCOM or even RMI are heavy, slow and painstaking to use that they were dropped by developers writing Web applications. It's above all because none could naturally cross the Firewall barrier... SOAP does not suffer this handicap, because it's based on HTTP. And this gives it one more advantage in terms of compliance with real standards!

So what is the true nature of Web Services? Client-server between applications, over the Net.

Since the technical side of things seems to have gotten off to a good start, we should be able to start believing the promises made by Web Services. The next step is to ask what their actual role will be.
The marketing departments of the leading industry players highlight the following objectives:

- To implement personalized information exchange between B2B partners.
- To offer and publish modular applications in a ready-to-use format.

The way is clear for a whole new class of applications. These will be inter-enterprise Web Services that will bring concrete form to the notion of the hyper-EAI: openness and urbanization of the information system on the scale of the Internet. What is more, the logic that holds true when it comes to opening applications up to the outside also holds true for projects dealing with the internal integration of applications (EAI).

It could become easier than before to integrate existing resources, thanks to SOAP, which has proven to be a fairly unintrusive technique (since it is only a lightweight RPC). We will see enterprises set up XML Hubs that will use SOAP to carry information in order to organize inter-application exchanges, whether inside the information system (EAI) or outwards (hyper-EAI or interface with partners' information systems).
Four years ago, we all saw the rush to develop Web sites on the Internet. Today, we will witness the rush to set up Web Services over the Internet.

*By Alain Lefebvre and Jean-Christophe Cimetiere, CEO, TechMetrix Research.*

 

# Introduction

The arrival of Web services promises to bring interesting changes in current Internet technologies. Application servers such as BEA WebLogic Server [2] or IBM WebSphere [3] will support this new type of application by integrating protocols such as SOAP. In the meantime, technical pioneers such as Iona [4], The Mind Electric [5], Shinka Technologies [6], CapeClear [7] and Idoox [8] have already taken the lead. Development tools that are totally Web service-oriented are appearing already, with Microsoft's .NET platform [9], for example.

The way that Internet applications are developed will be influenced by the new possibilities of offering modular services on the Web. As regards implementing these services, high level approaches are more than ever at the forefront. Up until now, the business object approach was reserved for early adopters but today is more widely promoted. The EJB specification has greatly contributed to raising awareness of concepts such as O/R mapping.

This White Paper does not offer a detailed description of technologies such as SOAP, UDDI or ebXML, but illustrates to what extent today's leading object-oriented component models are little adapted to new development styles. We will concentrate on two technical lines: support for the concept of Web service and support for the business object approach. We recommend certain technical alternatives and explain how we think Web services will impact on the implementation of inter-enterprise strategies on the Web.
We decided it would be useful to give a critical view of the three main component standards - i.e. COM, CORBA and EJB - in the Web services context. We will also touch on the new emerging standards and technologies such as .NET- CLR, SOAP, and JDO.

For further information on Web services technologies, you can consult the dedicated site on DeveloperWorks [10] and recent issues of TrendMarkers [11]. For the latest news, Webservices.org [12] is a good resource.

 

# What do Web services offer?

Web services mark a major step forward in the whole area of inter-enterprise cooperation. The notion of "extended enterprise" promoted during the 90s has found a true field of application.

Up until now, communication between enterprises on the Internet has been very limited. The advantages of HTML, which is at the root of the explosion of Web and e-commerce, may also be seen as its disadvantages. This multimedia document-oriented language gave the general public access to publishing tools, but also established a de facto standard which does not enable structured information to be exchanged.

Most Web sites developed with HTML are monolithic applications. The code, data, rules and presentation elements are mixed together. This approach does not enable structured data to be isolated from a data flow.

Web services, which are mainly based on XML, provide the solution. They make up a universal Application Program Interface (API) which does not require any protocol other than Internet protocols (HTTP over TCP/IP) and standardize the calling, exchange and organization of application services.

To illustrate the assets of Web services, let's take the example of an e-commerce application. A retail site, which sells PCs, wishes to integrate two functionalities into the offering. The site has no direct control over these two functionalities, which are shipping, provided by an express carrier, and credit scoring, provided by a credit institution.

Using current technology, the retail site does not have any satisfactory response for the two partners. One possible solution involves establishing Electronic Data Interchange (EDI) specific relations and then opting for specific development to integrate partners' modules on the platform. In any case, these solutions require heavy investment and much involvement from partners' IT departments. The retail site may solve this complex problem by using simple hypertext links between the retail site and partners' Web sites, but it will then lose control over the end customer.

Using Web services, the retail site can set up a solution for communicating with partners both quickly and easily. Using these standards, the site can dynamically call the shipping and credit scoring modules on the Internet. The result of the requests for these Web services can then be used by the retail site's application server.

Web services enable heterogeneous and distant systems to communicate by standardizing technical communication objects and reducing the cost of integration.

We believe that Web services are as important a stage in Internet development as HTML.

The direct result of this standardization of inter-enterprise exchanges will, in the short term, mean profound changes in the retail models on the Internet. From now on, thanks to technical and marketing alliances with partners, an enterprise can create and deploy product and service offerings with high added value.

 

In the area of financial services for example, all of the players are basing their strategies around two focus areas: creating multi-product and multi-supplier offerings (banking assurance, especially) and distributing these offerings over new channels (mass distribution, car manufacturers...).

Web services will help accelerate these strategic orientations. By supporting these open standards, financial service providers will be able to assemble their products more easily (assurance packages + consumer credit) and fit them dynamically into a multitude of sales contexts (included with the purchase of a car, in a cybermarket...).

The travel industry is heading in the same direction. Airlines, hotel chains, tour-operators and car-rental companies are gradually packaging their offerings to create turnkey products. If, for example, a client buys a plane ticket from Paris to Lyons, the salesperson can dynamically suggest a hotel and car for the same dates. This dynamic assembly of products and services gives the client real added value and provides enterprises with marketing advantages: lower costs for acquiring clients, sharing programs for customer loyalty, etc.

 

# Web services technologies

The general architecture which enables an enterprise to offer and use Web services is slowly beginning to take shape, thanks to initiatives taken by Internet leaders such as IBM, SUN, Microsoft, Oracle, HP, etc., who see it as an essential revolution. However, enterprises are still confronted by a multitude of technologies and different offerings. It is very easy to get lost in the myriad of offerings which includes SOAP, UDDI, WSDL, .NET, COM, EJB, CORBA, ebXML, BTP... However, these technological choices are strategic.

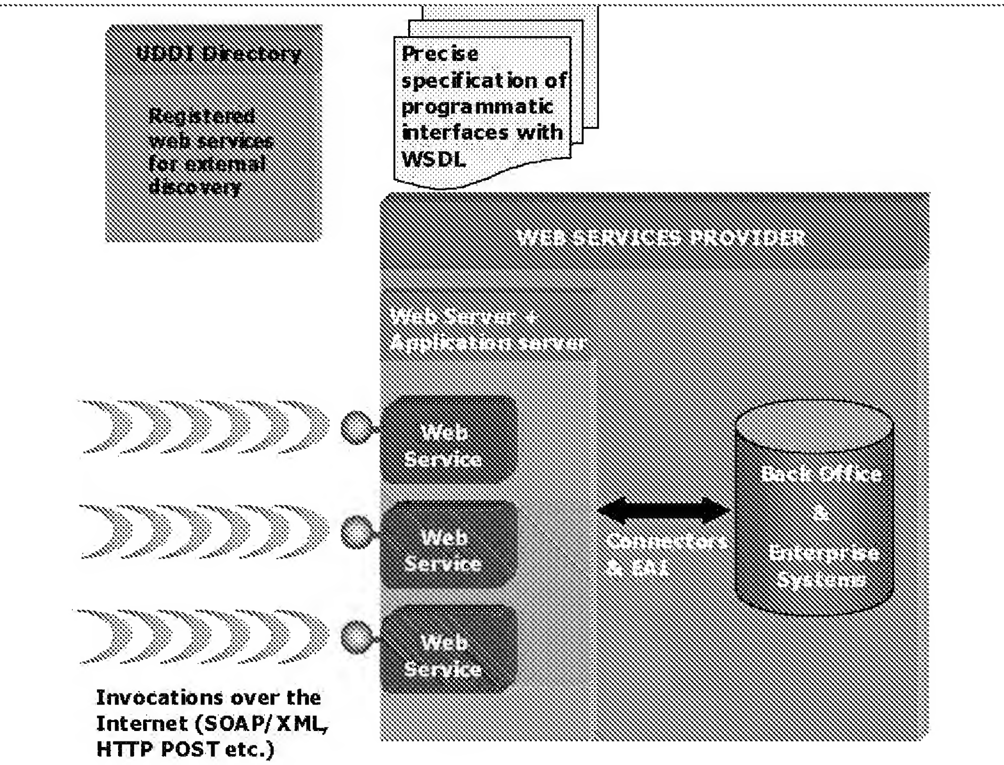


*Figure 1. Example of the architecture of a Web Service Provider.*

There is great freedom for implementing Web services internally. The only condition is to be able to present an interface to the outside world that is based on standard Web services technologies (fig. 1). There are, nonetheless, certain constraints on the quality of service: if you wish to be sure that partners and clients receive a flawless level of service, you will have to implement technologies which allow heavy workload and ensure reliable operation. You will also need to access your existing information system (database, ERP...).

In most cases, you will run your Web services within an application server. Over the coming months, you will witness a change in the existing application servers which will all have to integrate support for Web services.

 

The features of this technical infrastructure include:

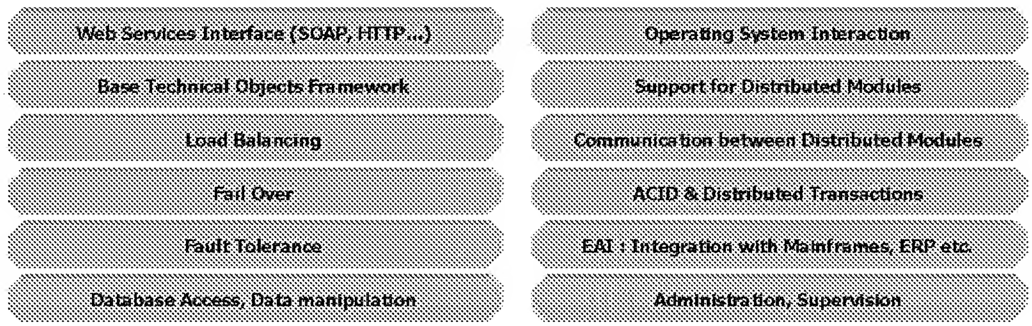


*Figure 2. Technical Infrastructure*

In terms of development, the increasing complexity of information systems, the need for rapid adaptability and integration of business concepts are all good reasons for adopting a high level approach which implements techniques of modeling and programming with business objects, as well as workflow tools which enable multi-agent collaborations to be managed both synchronously and asynchronously.

 

# Moving to the business objects approach

Object-oriented development techniques are at the center of many studies and reports. They mark a real revolution in the way software is built, and their implications are far from being completely mastered or even understood, which makes for fascinating developments in the academic and industrial worlds.

Integrating object-oriented techniques and infrastructure functionalities (fig. 2) is neither easy nor natural. Little by little, new concepts and scope for object programming directly related to this integration have appeared:

- Object persistence (O/R mapping, object database, hybrid database).
- Transaction on objects; Object Transaction Monitor (OTM).
- Distributed objects, remote method invocation.
- Object security, call filtering.
- Object query languages.
- Robust memory management (distributed garbage collection, reference counting).
- Threading and synchronization for objects.
- Object frameworks for the enterprise.
- And so on...

In recent years, several standards have emerged. The huge interest they have enjoyed may be explained by that fact that they responded to enterprises' highest hopes: need for interoperability between applications and for a lasting technology for unifying developments. Up until now, the industry has had three main rival models: COM (COM+/DCOM), CORBA and EJB. In our opinion, these technologies are not suited to the whole concept of Web service and are too limited in their support of object modeling. New standards are emerging, as we will see in the following sections.

 

# COM

## COM past and present

When Microsoft designed COM (Component Object Model), which served as a basis for DCOM (Distributed Component Object Model) and COM+, the main standard in the world of object languages was C++. COM was designed as a runtime specification. It used the exact same memory layout as C++ objects from Microsoft's compiler and solved the problem of calling remote objects. To this end, Microsoft integrated an object-oriented RPC system, based on an extension of Distributed Computing Environment (DCE), and a certain number of rules and specifications which take account of the constraints of a distributed object model: identification and destruction of unused distributed resources, specification of a system for error management, integration with a threading model, etc. Later, with COM+, Microsoft introduced transactional and security features stemming mainly from Microsoft Transaction Server (MTS).

Although based on C++, COM is a complete specification at binary level and nothing prevents other languages from producing or calling COM objects. Indeed, this sometimes happened and Microsoft is very much in favor of this idea of multi-language models.

Developing and deploying COM components is fairly complicated. JAVA took over from C++ in the world of object languages, thanks to its simplicity and flexibility. Luckily for Microsoft, it was able to integrate COM with Visual Basic. This enabled easier access to COM, although it used a new layer for some functionalities and, as a result, did not give access to the full potential of the model.

## Does COM/DCOM offer a distributed call protocol for Web services?

The answer to the above is no. The qualities of the inter-object communication protocol offered by DCOM are not questioned: we must give justice to important characteristics such as fault-tolerant reference counting and reentrance management, present since the first version of COM. However, Web services require a protocol which supports characteristics that DCOM does not have:

- Hyper-scalable protocol
- Protocol not blocked by firewalls that have standard configuration
- Use of HTTP
- Simple development and deployment

The huge failure of the model recommended by Microsoft (which consisted in using ActiveX clients within the Web browser communicating with a DCOM server) illustrates the problem of using DCOM on the Web.

# Does COM offer support that is suited to development using business objects?

Here again, COM has shown its limitations. Business object systems for Web service-type business applications need a rich runtime with, if possible, a real garbage collector, allowing object/relational mapping and high reflexivity capacities, giving access to object frameworks that may be re-used at high level, and so on. Clearly, COM and its primary runtime based on C++, even when enriched with extensions (Type libraries, IUnknown...), does not offer a suitable modern, flexible infrastructure.

# Tomorrow: .NET, C# and the Common Language Runtime

Microsoft is currently devoting considerable resources to developing an infrastructure that is suited to Web services. To this end, the Redmond giant has decided to offer a new model which replaces COM, the Common Language Run-time (CLR).

Microsoft's new C# language uses this object run-time to the full. It is one of the key elements of Microsoft's .NET strategy which integrates different technologies for implementing Web services (SOAP, XML...). However, the CLR is not designed exclusively for .NET - it may well become the global object model of Microsoft operating systems and applications.

We will not go into a detailed description here, although we will make the following comments:

- C# was designed under the management of Anders Hejlsberg, creator of Delphi and Visual J++.
- Microsoft has submitted C# to the ECMA for standardization.

- C# is an object-oriented language and is similar to Java in terms of the syntax and capacities (joke of the week: "*How do you convert Java code to C#? Change the file extension...*"). Nonetheless, it offers a large number of specific characteristics. For example, it offers all of its data types in the form of objects (contrary to Java and its primitive types). This considerably unifies programming. It also integrates the concept of events and delegation at language level.

- The CLR offers complete and integrated access to COM and .NET frameworks.

- The CLR is designed not only to be used by C#, but to be a "language-neutral" component model. Other development languages can produce a compiled code in Microsoft Intermediate Language (MSIL), the intermediate language used by C#. Various players in the world of languages have announced that they are working on this integration and the first tangible results are appearing. Theoretically, it will be possible to develop components based on the CLR in Eiffel, Smalltalk, Cobol, APL, etc. and to combine them without any restrictions within the .NET platform. This sophisticated capacity is the result of a great deal of work from Microsoft and compiler suppliers. The CLR enables modules created in different languages to be self-descriptive thanks to a common description standard (what Microsoft calls metadata). This type of common standard enables the CLR to greatly facilitate the integration of inter-language modules.

- However, this type of capacity has its limits. While it is possible to use different syntaxes (Eiffel, Smalltalk...) to produce CLR components, in the end, the CLR's semantics and capacities remain dominant, even when covered and adapted by a specific layer. The CLR was designed in cooperation with various compiler vendors in order to offer broad enough cover to support numerous languages, although it is not the ultimate solution. In practice, the languages integrated with the CLR are mostly modified versions. You don't use the Eiffel language itself, but a slightly modified version called Eiffel#. Similarly, it is not C++ that enables you to use .NET but an adapted version. In spite of these limitations, Microsoft seems to be able to offer an advanced solution in the area of integration between languages.

 

- Visual Studio .NET, currently in beta version, offers a rich environment for developing applications and Web services in C#. It makes it much easier to set up SOAP communication by offering modules (proxies) which take charge of the communication details (marshalling...). We were disappointed with its static system for proxy generation (WebServiceUtil) from a WSDL description. It requires a stage for generation of a proxy in the form of a DLL. This DLL is specific to each new Web service and must then be deployed on the platform running the program which calls the Web service (static glue approach).

- In addition to supporting the notion of Web service for setting up loosely coupled, heterogeneous distributed systems on the Web, .NET offers a replacement to DCOM for optimized communication between distributed objects. This replacement is called .NET Remoting and enables distributed objects programmed with .NET to communicate. This new model offers advanced characteristics such as security management, distributed garbage collecting, object passing by value or reference, object activation, transport over TCP or HTTP etc.

With CLR, C# and .NET, Microsoft is basing its future on a new model of technology that is much more advanced than COM. The new Microsoft frameworks (implementation of Web services, access to databases, graphical interface...) will be based on this model, as will the new development tools.

 

# CORBA

## A multi-language object standard

CORBA was created at the beginning of the 90s by the Object Management Group (OMG), an imposing consortium with over 800 members today. CORBA (or more generally OMA) is an ambitious initiative which covers a wide range of object technologies whilst keeping a remarkably unified view. The OMG's aim is to publish specifications – and they do. But the technical and marketing successes rub shoulders with some resounding failures. Among the successes, ORBs, which enable communication between distributed objects, were the spearhead for CORBA and found support in certain sectors of the industry; they led to numerous implementations from companies such as Iona, Borland, SUN, IBM, DEC, AT&T, etc. Certain services such as OTS (Object Transaction Service) are today at the heart of transactional object monitors used by application servers. Other specifications such as MOF (Meta Object Facility) will be at the forefront of the wave of high-level model construction and transformation technologies.

One of the essential features of CORBA is that it sits at a high level of abstraction; for example, the various frameworks and services are specified not using a particular programming language, but via an IDL (Interface Description Language). The binding with a specific programming language (in other words, the way in which the IDL concepts are projected onto the concepts of a particular language) is described separately and does not interfere with central specifications.

 

# Does CORBA offer an infrastructure for distributed calls that can be used for Web services?

As with COM, the answer is no. For a long time, CORBA supporters saw CORBA as a universal communication bus between components distributed across the four corners of the world. This vision of "Intergalactic Distributed Objects," described in the classic *The Essential Distributed Objects Survival Guide* by Robert Orfali, Dan Harkey & Jeri Edwards, influenced numerous developments and initiatives in the industry but did not lead to anything concrete.

Today, some aspects of this vision may well become a reality, but this will mostly likely be through Web services and not CORBA.

## A protocol not suited to the Web

The Internet Inter-ORB Protocol (IIOP), which is at the heart of CORBA objects communication, does not offer the characteristics required to access Web services. While IIOP is a high performance protocol, it does not have the required scalability required for Web services. In addition, it does not pass firewalls by default and this is essential for setting up Web services. It is possible to overcome this difficulty in one of two ways, although both are unsatisfactory in the Web services context:

- Configure the various firewalls to enable IIOP to pass. This solution is heavy and comes up against important technical and organizational difficulties. Among the technical difficulties are support for the CORBA concept of server location transparency and the support for the generalized object communication model in which each object can be a client and a server. OMG itself explains this last point as follows: "*Because standard CORBA connections carry invocations only one way, a callback typically requires the establishing of a second TCP connection for this traffic heading in the other direction, which is a no-no to virtually every firewall in existence.*"

- Use HTTP tunneling which consists in encapsulating IIOP messages in HTTP frames This technique, offered by certain companies such as Sybase for its EA Server and Borland with Gatekeeper, is still immature and the chances of it being generalized are slim. One of the main difficulties is that it generally requires using the same CORBA ORB with the various partners wishing to communicate. This is entirely unsuitable in the context of Web services.

 

The OMG has adopted a specification dealing with IIOP's traversal of firewalls. This document includes the very useful, bi-directional GIOP communication capacity and describes how firewalls can have a reasonable control over CORBA calls. Unfortunately, this specification presents various problems, which the OMG itself has clearly indicated.

> *"The current specification for CORBA traversal of firewalls has a number of errors that are beyond the scope of an RTF* [Revision Task Force]*. These errors have prevented the existence of a consensus approach to mechanisms by which firewalls can control IIOP network traffic and enable controlled use from the Internet of selected CORBA-based application."*
>
> *Object Management Group, orbos/00-09-20*

Faced with this situation, the OMG relaunched a complete specification process. The new RFP (Request For Proposal) "CORBA Firewall Traversal" was published in September 2000 and a preliminary proposal was recently submitted to the OMG by a group consisting of Borland, IBM, Sun, IONA and XTRADYNE. The RFP phase should be completed by June 2001. We are therefore at the beginning of the process. The preliminary proposal offers good perspectives for integration between CORBA and firewalls, although does require that the firewalls support this future specification. In other words, new generation firewalls are required.

It also contains certain useful aspects on the techniques which may be implemented with the current firewalls for CORBA communication.
If it succeeds, this initiative will be very useful, as it will enable the IIOP flow to be controlled intelligently. But it will take time for it to become standard.

Today, IIOP seems to be unsuitable for Web services and for the moment, it is unlikely that a Web service friendly GIOP (General Inter ORB Protocol) specialization will be developed.

The OMG has published an RFP entitled "CORBA/SOAP" (ref. orbos/00-09-07), calling for proposals for supporting CORBA semantics on SOAP, and access to CORBA services from SOAP clients. It will be interesting to see the responses to this RFP.

SOAP is designed to circulate on all transport protocols and in particular on HTTP (SOAP 1.1 is also implemented on SMTP and is open to other protocols) and therefore passes through traditional firewalls which enable HTTP streams to pass through port 80. This may lead to security flaws. It is possible that the SOAP specification will take this problem into account in the future. Nevertheless, techniques do exist today which enable secure SOAP Web services to be developed. Contrary to IIOP, JRMP and DCOM, which are binary protocols, SOAP uses XML and is therefore easier to interpret on the fly. In addition, SOAP messages are extremely verbose and are, in a limited sense, self-describing. This enables filters to be built more easily and security rules to be applied. The securing of a SOAP Web service must also be based on current security techniques used in Web applications. The fact that the requests are no longer sent by a Web browser but by an application poses certain security. In the end, setting up security is up to the Web service developer, although it is possible.

 

## The problem of static glue

An application which calls a Web service must be easily developed and deployed and must require the fewest possible technical contacts between the developer of the client application and the Web service provider. Ideally, it would be best to avoid having to download, generate or install specific components on the client machine for each Web service.

On this point, CORBA makes things difficult as, for most of the time, it is based on static glue technologies: with standard market tools we must generate binary code packages and deploy them on client stations when setting up new distributed services or when modifying the interfaces of these services. Certainly, there is an API in CORBA, the Dynamic Invocation Interface (DII), which can be used for calling distributed services without using the static glue. DII is an essential building block for using dynamic glue (a dynamic glue technology can be implemented above the DII), although it is not a transparent dynamic glue technology itself. As a result, programming calls with DII becomes fairly complicated. When we add the complexity of CORBA development, we can see that we are far from the flexibility and super-easy connectivity required by Web services.

Our experience of static glue and dynamic glue systems is that they are a major technical factor in the success of a project. The bigger the project, the more likely it is to crumble under the weight and lack of flexibility of the static glue. And this is even more true in the case of worldwide deployment on the Web. Ron Resnick dealt with this question as early as 1996 in [13].

> *"Not until the dominant usage of CORBA objects is through the DII rather than static precompiled stubs could one claim that CORBA objects are ready for public consumption on the Internet. Of course, since the Internet has not been much used programmatically, this absence has not been much noted."*
>
> *Bringing Distributed Objects to the World Wide Web, Ron I. Resnick.*
> *http://hegel.ittc.ukans.edu/projects/corba/postscript/javacorb.html*

The arrival of programmatic calls on the Web, i.e. the arrival of Web services, highlights the limitations of static glue. Unfortunately, very few CORBA products take advantage of DII to offer developers the advantages of a dynamic system. One of the rare exceptions is IDLScript, which enables interactive calling of distributed CORBA objects (incidentally, it is a rare pearl).

Using Web-services oriented technologies such as SOAP does not necessarily mean using dynamic glue. We recommend paying close attention to this when choosing Web service technologies and the associated development and deployment tools. A dynamic technology enables systems to evolve more easily, to support new requirements with minimal effort. In addition, this considerably increases development productivity by facilitating prototyping and experimentation.

The availability of dynamic technologies for Web services partly depends on the development languages used on the server and on the client. The more dynamic the language, the more capable it is of accepting a dynamic Web service technology. We cannot expect miracles from environments that are mainly static and not very reflexive such as C++. Java is more flexible

 

and improves gradually with each new JDK version. A maximum of power and flexibility is offered by the highly dynamic languages (Smalltalk, Python, Perl...)

## CORBA's failures on the Internet

CORBA's lack of adaptability to the Web is illustrated by the various failed attempts to promote CORBA as the communication model between client applications and enterprise servers on the Web.

When NetScape integrated CORBA technology within its Web browser a few years ago, tens of millions of stations had the possibility of connecting to distributed CORBA technologies. But what happened? Nothing! And Iona fell victim to the same fate with its OrbixWeb technology.

On another topic, the market failure of BEA's M3 CORBA application server (now renamed WebLogic Enterprise, and not to be confused with WebLogic Server which enjoys phenomenal success) illustrated how a vision can sometimes come up against technical constraints.
We could also mention the lack of market interest in the Borland Application Server, which has CORBA at the center of its technical strategy.

# CORBA as a Web services directory?

Are the CORBA Interface Repository and Trading Object and Licensing services adapted to the context of Web services?

These CORBA elements are used to describe the interfaces of distributed objects in a precise manner and to locate them using interrogation criteria or to manage billing linked to the use of these objects (payment of object providers). These functionalities promised super directories of objects spread worldwide. In fact, they are very little used, or almost never implemented, in the case of Licensing services. Among other things, they came up against the fact that they offered CORBA access interfaces (highly logical) and therefore were hit with all of the problems described above. New better-adapted candidates such as UDDI appeared. Note, however, that these Web service directory technologies are still very immature and the domain is far from stable. We remain hopeful that it will be possible to reuse some of the very useful research and design carried out in the field of CORBA services regarding these complex subjects.

# Where can CORBA's distributed capacities be used?

The problem of calling Web services is very specific: the calls must pass via HTTP (the protocol of the Web), the system must be scalable etc. But there are numerous situations where we wish to implement a distributed system with other characteristics: high performance calling, frequent and mass data exchange, integration with existing CORBA systems, advanced transactional semantics, reference semantics etc. In these situations CORBA may be adapted.

## CORBA, for high-performance, inter-language communication

Clearly, for communication by distributed objects that are both high performance (unlike SOAP) and inter-language, CORBA remains king and will stay that way for a long time yet. It will also be required for accessing enterprise systems, internal or external to an organization, already offering a CORBA interface, which is the case in certain fields such as the stock market or telecoms. CORBA is also especially suited to distributed calls which must spread a transactional context.

## The limitations of CORBA's multi-language model

CORBA sits at a very high abstraction level to avoid being restricted to a particular programming language. This enables objects written in different languages to communicate on a CORBA bus. To do this, CORBA could only be based on concepts that can be translated into various languages (C++, Smalltalk, Cobol etc.), and is therefore limited to the lowest common denominator. CORBA is generic, but there is a price to pay: it is extremely complex and very limited. For example, passing objects by value, while very useful, only became a feature very recently with the OBV specification and, for the moment, can only be used with a restricted number of languages. Another example: CORBA does not provide a satisfying solution to the problem of managing distributed memory (distributed garbage collecting, for example), however fundamental. It is not easy to specify an abstract mechanism for managing inter-language memory when these languages function differently in this aspect.

## CORBA faced with optimized single-language models

If you want to enable distant objects written in the same language to communicate, but do not require a Web services approach, it is more suitable to use the native model offered by the language in question as it is not limited by a multi-language model like CORBA. For example, Java RMI is easier to use and more powerful in most contexts than a CORBA solution: distributed garbage collecting, passage of Java objects, remote loading of the code, does not use IDL, etc. Another example is the Objective-C distributed objects that we find in the new Mac OS X and in Linux/GNUstep, which offer flexibility and integration with a language incomparable with a CORBA solution. For C++, on the other hand, CORBA is often the most suitable solution. Unfortunately, CORBA/C++ mapping has not kept up with ANSI C++ language standardization efforts (Standard Template Library etc.)

 

# Is CORBA suited to business object development?

## The CORBA business object view

CORBA is not only destined for communication between distributed objects but covers a large section of the fields of application of object technology. CORBA specifies object persistence services, transaction services, services for managing associations between objects with support of the various cardinalities and referential integrity rules, object query language, object collections, etc. The recent CORBA component specification aims to integrate all of these services and to offer an equivalent to EJBs, although this time in a multi-language world. We may then wish to adopt CORBA technology to develop business objects that are internal to Web services. The interface of each class object would be described in IDL and we would use the various CORBA services (persistence relationship, linking...) as a general infrastructure.

## Beware!

We can admit straightaway that such an approach is almost bound to fail: too heavy, damaging complexity, limited semantics and certain services unavailable. This use of CORBA was widely rejected in practice and we would not be keen to recommend it. We also think that the arrival of the CORBA components specification will not solve the problem. On the contrary, the heaviness and complexity (thousands of pages...), of this specification (currently being finalized) emphasize the limitations of CORBA in the general development of business components. It is likely that if the CORBA Components specification is finalized – which is far from certain, due to the state of the finalization task force process and OMG's deadlines – it will be quite incomplete and not widely implemented by the players in the CORBA world. The only exception may be In Java, since it has very strong design similarities with the EJB specification. But even this last point is in jeopardy given the important changes underway to the EJB specification.

 

# EJB

## The EJB vision

Enterprise JavaBeans is an imposing specification (more than 500 pages in the last draft), promoted by SUN as the component model for the J2EE platform. The current versions of the specification (1.0 and 1.1) are implemented by various vendors of application servers such as BEA, IBM, Oracle, SUN, HP-Bluestone, Brokat, etc. Open Source implementations are also included. Version 2.0 of the EJB specification was still only a draft at the time of writing (June 2001).

EJBs are meant to be a type of "super Java object", integrating object life cycle management, security management for method calls, Object/Relational mapping, transaction management (triggering, coordination) and distributed access. Ideally, these components can be used on all servers that offer an EJB infrastructure, as long as they're implemented without using proprietary extensions; this is the "reusable packaged components" idea.

EJB technology was originally designed to enable remote access to components in a secure and transactional context. Also, it is particularly suitable for implementing coarse-grained components, which fits in with the granularity of Web services. In addition, EJBs are designed to transparently support execution in environments that implement load balancing. The integration of JMS in the EJB 2.0 draft allows for a true modeling workflow of the processes implementing EJBs.

Unfortunately, a number of significant technical problems have appeared to darken the horizon.

# Does EJB technology offer a distributed calling protocol that can be used for Web services?

## Protocols that are not suited to the Web

The protocol specified by EJB is RMI, or optionally, RMI over IIOP (implementations of this option are fairly immature at the moment). We come up against the same type of problems as with DCOM and CORBA, which we will not detail here. RMI and RMI over IIOP are not suited to the concept of Web service: they do not cross standard firewalls, they are not designed to use HTTP, they don't offer the required scalability, etc.

An excellent article written by Billy Newport [14] clearly presents the elements required to choose between SOAP and RMI or RMI over IIOP. It would appear to show why RMI and RMI over IIOP will not be used in a Web services context.

## Limited interoperability

Managing interoperability between heterogeneous platforms is at the heart of the Web services revolution. In this domain, EJB technology does not really provide the solution. The current main EJB implementations do not offer satisfactory direct interoperability with non-Java platforms. Even worse, interoperability between different EJB implementations does not exist or else requires operation in degraded mode.

The possibility of communicating between EJBs deployed on heterogeneous application servers (for example, invoking a WebLogic EJB from a WebSphere server) or from CORBA clients (for example, a C++ client invokes an EJB) is theoretically based on RMI/IIOP. The idea is as follows: "IIOP, the standard CORBA communication protocol, enables a vast range of heterogeneous platforms to interoperate. RMI is the EJB distributed calling method. Support for RMI over IIOP by EJBs should enable J2EE and CORBA platforms to interoperate."

The idea is attractive but is often a non-starter in practice. Our experience has shown that, in numerous cases, it was impossible to communicate with a given application server if this server's ORB is not used on the client side (for example, certain vendors have announced support for RMI/IIOP while in fact they only support a "proprietary" version of RMI/IIOP!)

 

Certain application servers are now beginning to support interoperability. In general, this remains limited to access from J2EE or C++ clients. CORBA/IIOP tests carried out internally by vendors are most often limited to Java and C++ client strains. In addition RMI/IIOP requires that ORB support CORBA 2.3. This support is not very widespread in the CORBA world, although the situation is gradually improving especially with the OMG publication of CORBA Language Mappings which supports version 2.3 for C++, Java, Lisp, PL/I and Python.

When a degree of interoperability is supported, the technical difficulties surrounding security management, transactions, load balancing and failover start to appear. See the article by Billy Newport on this subject [15].

> *"So, if we use RMI/IIOP, we lose clustering and security. We may lose transactional support depending on whether the OTS implementations are interoperable. The question has got to be asked when I lose these 2 or 3 things, especially security; can it really be called interoperable anymore? Sun is pushing for interoperability in EJB 2.0. But, it's difficult to see how they can achieve this in any 'real' sense unless these issues are addressed some-how. As it stands, if RMI/IIOP is the future of J2EE interoperability then the future looks bleak right now. Until Sun (it owns the J2EE standards process) addresses just how exactly RMI/IIOP is going to be standardized to the point where interoperability can be assured then any promises of interoperability will be based on very unrealistic scenarios (example, no security!)."*
>
> From *RMI/IIOP, nice idea but the reality is turning out to be different, by Billy Newport.*
> *http://theserverside.com/resources/article.jsp?l=RMI-IIOP*

It is important to note that the various problems discussed here depend greatly on the application server used. For example, BEA WebLogic is worse off than Borland Application Server where CORBA interoperability is concerned. This server enables advanced clustering possibilities to be retained, notably thanks to the possibility of managing load balancing on the server side without imposing the use of proprietary smart stubs on the client side.

In the rare cases where EJB/CORBA interoperability is possible, there is little interest in presenting EJBs as Web services, as we explained in the section on CORBA/IIOP.

# EJB vs. SOAP

We tend to hear the following two main arguments being voiced:

1) SOAP is a technology which enables distributed modules to communicate. EJB is a component technology. As a result, the two are not in competition and should not be compared.

2) SUN had added support for IIOP to the EJB specification. Likewise, we just have to wait for the addition of SOAP support for calling EJBs so that they can natively become Web services and thus solve the problem.

As regards the first point, it is true that the entire EJB specification cannot be compared with SOAP. On the other hand, it is important to remember that EJB is a **distributed** component

 

model and that this distributed aspect is one of the specification's main features and purposes. It is useful to attempt to understand the differences between SOAP and EJB as regards distribution and to evaluate the consequences of these differences in various contexts, in particular Web services.

As far as the second point is concerned (potential adoption of SOAP as a native protocol for calling EJBs) we think that this is rather misguided. SOAP semantics are too different from EJB calling semantics to enable SOAP to be used as a native protocol for calling EJBs. We must remember that SOAP is infinitely more primary than RMI. The simple fact that SOAP does not support passing by reference or the fact that it does not maintains stateful remote connections is enough, as far as we are concerned, to render impossible native support for the EJB calling semantics on SOAP. Of course, we can overcome this difficulty by using the extension system included in SOAP. But this amounts to creating a sort of "extended SOAP" and means that the interoperability with the SOAP standard is lost. There are also pure design arguments as regards the interfaces for programmatic interactions on the Web.

Steve Vinoski summarizes the situation in [16].

> "Many would-be Web Services developers want to expose, as web services, business logic already encapsulated within existing EJBs or CORBA objects. This in itself is neither unusual nor undesirable. After all, both EJB and CORBA are themselves used to wrap existing business logic (perhaps written as Java or C++ objects) and export it to other applications. The key word here, however, is "wrap."
> Exactly what does it mean to "wrap" an EJB or CORBA object and expose it as a web service? The answer lies in the realm of application-level protocols, and it involves the important issues of granularity and coupling.
> [...]
> I run into someone almost every single day who doesn't even realize that these problems must be addressed for Web Services to be used for enterprise applications. Because of all the Web Services hype that currently exists, they believe that Web Services is simple magic that allows you to effortlessly turn all your CORBA objects and EJBs into Web Services [...] Application-level protocols, granularity, coupling, and levels of abstraction all matter. [...] you can't meaningfully directly expose a CORBA object or EJB as a Web Service[...] Implementing practical Web Services requires wrapping your internal infrastructure middleware (your EJBs, your CORBA objects, and your messaging systems that run your business) into meaningful workflows that provide the necessary business processes at the appropriate levels of abstraction."
>
> In "Application-Level Protocols", by Steve Vinoski, Chief Architect, Iona Technologies.
> http://www.iona.com/sphere/is0601.html#stevev

We believe that it may be useful to use EJB technology (or CORBA) within an application server, not to present Web services to the outside world via the EJB distributed calling model (nor even just to wrap the EJBs behind a simple SOAP facade, even if this is technically possible in certain conditions, cf. [17]), but as business components implemented by top-level processes defined in the Web services layer.

Nonetheless, this leads to the question of how useful EJB technology is to Web services as it loses one of its main functions: controlled access to the sever from remote clients. Before

 

making any rash decision to use EJB technology, ask yourself, in light of what we have just discussed, whether standard Java objects would not do.

# EJB for business object modeling within Web services

## EJB, the quest for location transparency

Entity Beans are EJBs destined to be used to represent persistent business objects. But note that the "super objects" we presented earlier are in fact "super sub-objects," offering additional functionalities but suffering from various limitations and complexities compared with standard Java objects. These limitations are not gratuitous, however; they authorize high transparency for locating objects used locally or remotely. This allows the underlying system to distribute the EJBs as it wishes, according to a load balancing logic, for example. The idea of generalized location transparency is attractive, as it would be practical to be able to develop distributed systems without having to worry about the distributed aspects. As it is not technically possible to use distributed components like local components, the designers of the EJB specification suggest doing the exact opposite: using all of the components, including local, as if they were distributed objects and thus offering generalized location transparency and enabling EJBs to be distributed objects *by nature*.

As a result, EJB development is subject to drastic rules, high complexity and limitations. The aim of EJBs is to facilitate development by concealing complexity. The result is far from this. This is certainly a problem of immaturity and also intrinsic complexity linked to the technical approach adopted.

## Complex EJB development

The tedious configuration of XML deployment descriptors, packaging of classes and deployment within the application server quickly become much too heavy for our taste. The fact that it is based on meta-information stored as external XML files and generating all sorts of implementation classes (static glue approach) tends to drown business objects in an ocean of technical modules, whose interactions are of varying obscurity and very little documented. At the end, we often find ourselves with error messages that are difficult to understand without having in-depth knowledge of the container's internal machinery, and this is exactly what the EJBs are supposed to free us from!

Entity Beans aim to allow the modeling of business objects and database persistence of these objects. But heavy and limited Entity Beans are, in our opinion, not suited for their task. If you have the chance to use a good O/R mapping tool or a good object-oriented database for your developments, you may well be disappointed by the Entity Beans approach.

 

## A static glue approach

In order to ensure distribution functionalities, transactional monitor and, in certain cases, object persistence, current EJB containers generate numerous service objects which encapsulate business objects provided by the developers. These service objects, which are a type of glue, intercept different events (in particular invocations of business object methods) in order to ensure their functionality: transmission of calls on the network, coordination with the transaction manager or implementing persistence. Current EJB technology uses a specific glue for each business class. This glue should be generated using a tool specific to each container, each time something is added to a business class or if it is modified. The generated static glue is then compiled and integrated with the application.

This approach is called static, as opposed to dynamic. Dynamic glue is capable of adapting to any type of business class without code generation.

As we explained when talking about CORBA, static glue poses many problems which are amplified when it comes to deploying services on the Web.

| | Dynamic glue | Static glue |
|---|---|---|
| **Glue generation** | No generation | Use of specific generation tools |
| **Deployment** | No deployment stage for the glue, either on the server side or on the client side | Deployment of the glue (server and client side for the glue of distributed objects) each time there is a change in the interfaces of the modules |
| **Dynamic invocations without static glue** | Transparent, use of the model of dynamic invocation native to the language | Not supported or use of a specific DII-type API |
| **Glue management** | No management | Several generated modules to manage (where to store them, how to provide different versions...) |
| **Support for fast prototyping** | Yes | No (see previous points) |
| **Support for evolution and correction in short phases** | Yes | No as this requires a client side deployment stage |
| **Performances** | In some cases, slight performance overhead from dynamic glue during calls | Addition of new glue classes for each new business class which leads to increased weight of the run-time memory footprint. |

Generally speaking, the static glue approach is found in common EJB implementations. Some implementations, such as JBoss, integrate more dynamic elements.

 

## Pay attention to performances

Using EJB Entity Beans as elements for object modeling is one of the most sure ways of severely degrading application performances. In particular, once the object graph becomes a little complex, access to databases is not minimized intelligently and the number of "pointless" accesses increases. Aside from problems linked to accessing the base, performances suffer from the fact that calls to Entity Beans are considered as remote calls. The specification [18] indicates that calling EJBs is too costly to generally support fine-grained objects.

> *"Every method call to an entity object via the remote and home interface is potentially a remote call. Even if the calling and called entity bean are collocated in the same JVM, that call must go through the container, which must create copies of all the parameters that are passed through the interface by value (i.e., all parameters that do not extend the java.rmi.Remote interface). The container is also required to check security and apply the declarative transaction attribute on the inter-component calls. The overhead of an inter-component call will likely be prohibitive for object interaction that are too fine-grained."*
>
> *Enterprise JavaBeans Specification, version 1.1*

Techniques exist to overcome these problems to a certain extent, as illustrated in [19]. These techniques remain partial and heavy and require contortions in the design of business objects.

Performances vary depending on the EJB container used but, in general, we have noted that vendors of EJB application servers themselves warn customers of the performances which come from using Entity Beans.

## Difficulties of the object-oriented approach using EJBs

Other technical elements are added to the difficulties discussed previously (see [20] and [21] for a discussion on some of the issues listed below):

- Difficult support for re-entrance, capacity often used in object programming.
- Various incompatibilities with the standard Java object model.
- No support for inheritance at component level.
- Difficult support for polymorphism.
- No adequate support for the concept of relationships between business objects.
- Very little support for querying.

In the end, it is misguided to hope to use EJBs to support business object modeling. We recommend reserving their use for RMI interactions for large-grained distributed components, and only if you need the transactional model and security model that are specific to EJBs. When choosing an application server it is important to verify that the advanced capacities of the application server (load balancing, fail-over, fault-tolerance etc.) do not impose the use of EJBs.

 

# EJB 2.0

The EJB 2.0 specification was still in the form of a draft at the time of writing (June 2001). The latest model of the draft was made public at the end of April 2001. It is a radical evolution in the notion of EJBs and introduces the concept of a local EJB component. We consider it to be a very positive move.

The previous generation EJB 2.0 draft specified the use of dependent objects so as to provide better support for object modeling. Relationships between EJBs were also supported, as was the modular notion of persistence manager. But everything in Entity Beans was still designed to make them distributed objects. This approach culminated with the publication in October 2000 of "*EJB 2.0 proposed final Draft*." With this draft, the specification moved a little closer towards a dead-end.

Luckily, the specification designers decided to radically change their approach. The new draft completely abandons the idea of dependent objects found in the previous draft (the specification of a modular persistence manager, probably considered as an added factor of complexity, was also removed). The big news is that the new draft introduces local EJBs. One of the fundamental errors of the EJB approach (generalized location transparency) is therefore recognized and, partly, repaired.

The new EJB 2.0 draft improves support for the object-oriented approach. Among the improvements are support for inter-object relations at local EJB level; implementation of interactions with the database is better supported with the introduction of a query language. Rules enabling the object lifecycle to be managed are introduced (cascading delete). Very important optimizations, such as partial loading of object graphs in the memory, are better supported.

Introducing local EJBs eliminates certain programming constraints that rendered the previous approach too heavy most of the time. In particular, there is no longer any need to manage a potential java.rmi.RemoteException for each call. It is also possible to pass a Java object by reference and thus share the same Java object from several local EJBs.

This new model is better suited to fine-grained objects than the previous one, but still has a long way to go.
We will have to wait for the publication of the final version of the 2.0 specification to know if this new approach will be adopted, as it highly likely.

Having said that, local EJBs open up a whole new range of problems. It is not easy to introduce such a profound design change into an already complex specification. New questions are posed and may lead to a new revision before adoption; one example is the question of relations between Entity Beans that are in different deployment units (jar).

 

## Conclusion: EJB and object modeling

The first official EJB specification was published at the beginning of 1998. At the time of writing this white paper (June 2001) EJB technology is three years old and doesn't offer anything even vaguely resembling a satisfactory environment for business object modeling. Two possible scenarios may arise in the future: in the first, the specification evolves radically , succeeds in offering a true object persistence framework, and is followed by the arrival of stable and high performance implementations. In the second scenario, EJB technology gets caught up in internal technical problems and hands over to other approaches for support for object modeling.

The likely arrival of the concept of a local interface for EJBs in the next 2.0 specification would appear to be good news, and leaves us with some hope that all is not lost. We must now wait for version 2.0 adoption and the arrival of complete implementations to be able to judge it on actual evidence.
But even in the best case scenario – successful radical evolution – we think we will have to wait at least two or three years before having a suitable technology that is sufficiently mature.

As a result, we must look towards other technologies for implementing persistent business objects in Java. This will be dealt with in the following section.

 

# What technologies can be used for developing Web services in Java?

## J2EE

Enterprise Java Bean is presented by Sun as the model of business component within the J2EE. But, as if we claim, EJBs are suited neither to object development nor to deployment on the Web, does this mean that we will have to reject J2EE as a whole - or even Java?

Fortunately not! Java is a good language for object-oriented programming and we believe that J2EE will be one of the best platforms for developing and deploying Web services. Here at Orchestra Networks, we have chosen a J2EE infrastructure.
To understand why, we have to note the relationship that exists between EJB and J2EE. It can be represented as follows:

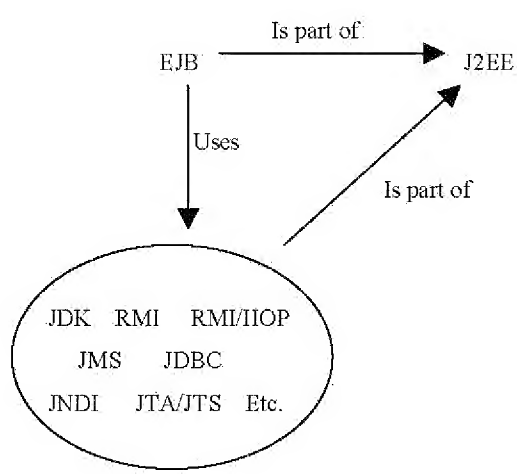


*Figure 3. J2EE / EJB relationship*

This schema shows that J2EE is made up of numerous frameworks ("is part of" relations). EJB is based on the other J2EE frameworks as illustrated in the "uses" relation. EJB cannot therefore be used without certain J2EE elements. But the situation is not symmetrical.
The other J2EE frameworks are not based on EJB and don't use it at all: there is, therefore, no functional dependence. This means that J2EE can be used without EJB; this is the case of the vast majority of existing J2EE applications. Sun's presentation of EJBs as a central element should only be seen as a *recommendation for application architectures*. It follows from the present discussion that if Sun decides to abandon EJB technology or part of it (for example, Entity Beans) this will not have an impact on other J2EE elements.

 

It has already been established that most J2EE application servers will offer (or already offer) extensions to their application servers that support new Web services standards and, above all, SOAP. Of course, the question of standardizing these extensions at J2EE level arises. For instance, a JSR (Java Specification Request) [22] was recently submitted by IBM as part of the Java Community Process.

> *"Much in the same way that Servlets tied together a set of concepts like cookies and HttpSession, and EJB tied together RMI, JTA/JTS, JDBC, etc. with a programming model and runtime model, we view this JSR doing the same for implementing and using web services."*
>
> *JSR-000109 Implementing Enterprise Web Services, IBM Corporation.*
> *www.j2ee.com/aboutJava/communityprocess/jsr/jsr_109_implement.html*

In the meantime, we recommend using J2EE for developing Web services in Java, although we advise not using EJBs as a distributed access system to Web services or as a persistence tool for business objects.

# HTTP

Communication with a Web service must use Web protocols, in other words HTTP (& HTTPS). In our opinion, today this is what best defines Web services. Thus, we will use HTTP for invoking a Web service and not DCOM, IIOP (CORBA, EJB) or even JRMP (RMI, EJB). Certain groups are studying the possibility of using other Web protocols such as FTP or SMTP for transporting invocations to Web services.

As far as the distributed model for accessing Web services is concerned, you must choose a technology that is based on HTTP. SOAP seems to be well on the way to becoming the main standard in the domain; the future W3C XP recommendation will doubtless also play an important role. Other building blocks such as WSDL or UDDI may be used to complete the picture. Frameworks such as ebXML, BTP or RosettaNet, if adopted, will allow programmatic interaction between enterprises to be better structured. These emerging standards are not yet widespread. They remain immature and are in need of improvement, although they are evolving rapidly. Enterprises can already begin experimenting with SOAP, for which certain implementations are available in Java.

The young age of SOAP, WSDL and the like does not prevent enterprises from implementing Web services today so as to interact with partners who do not yet support SOAP on their platform. Almost all platforms have APIs for sending and receiving HTTP requests, and can pass simple parameters, or even complex XML documents. An enterprise can therefore easily set up simple programming interactions on the Web by standardizing and structuring POST HTTP calls with partners, whatever their platform.

Java offers the required APIs for sending and receiving HTTP streams. For example, we can use the `java.net` package for sending queries. The `HttpURLConnection` standard class offers some useful functionalities and it is also possible to use third-party frameworks if required. Servlets allow queries to be received.

 

For more sophisticated requirements you can use XML-RPC, SOAP's ancestor, which is also available on numerous platforms.

The SoapRPC web site [23] site offers numerous references to help you find an XML-RPC or SOAP implementation that is adapted to your platform.

# XML

In numerous cases, using POST HTTP with simple parameter passing is enough. In more sophisticated cases, XML is carving its niche as the standard format for the data exchange that takes place when Web services are invoked. At development level, it is possible, and sometimes preferable, to move away from XML representation and to work uniquely with the structures of one's own programming language, relying on a mechanism for automated translation, at runtime, into XML. In other instances, it is preferable to explicitly model the data exchanged in XML and to manipulate this data at XML level.

In all cases we will use dedicated APIs. For Java, numerous APIs are emerging (see [24]): JAXP, JAXB, JAXM, JAX/RPC... For the moment, high level APIs are not yet finalized. On the other hand, low level APIs enabling parsing or construction of XML documents do exist and there are some solid implementations. In this domain, the standard API adopted by Sun for Java is JAXP (Java API for XML Parsing). We may also need tools for modeling XML documents. Finally, changes in the XML world itself must also be taken into account.

For our part, we are using the recent XML Schema recommendation instead of traditional DTDs to define the format of XML documents. It is more powerful and is increasingly supported by tools and frameworks. However, certain aspects of support remain immature, notably namespaces and validation. We must therefore remain pragmatic and favor simplicity for modeling XML Schemas. For XML modeling, we use the XML Spy graphic workshop. Finally, as regards the API, we consider that JAXP, which is based on the classic SAX and DOM, is not sufficiently suited to Java. This makes programming unnecessarily heavy. If the SAX and DOM APIs are today almost essential to Java it is because of the maturity of the implementations. We do not believe that they represent the future of APIs for directly manipulating XML in Java. In our opinion, the current most promising project is JDOM [25]. This open source project is currently in development and we evaluated it in the context of a real, non-trivial B2B integration scenario. We noticed that the API provided is better than JAXP. This reduces the amount of coding and produces a clearer code. The current beta version is very convincing.

 

# Persistence framework and transactional object monitor

## A domain in turmoil

J2EE offers a good basic infrastructure for setting up business objects. It lacks a good persistence framework and a good object transactional monitor. The two concepts are actually closely linked. The persistence framework should be capable of running in a transactional context (ACID transactions) and therefore be based on an object transactional monitor. It should also offer querying possibilities, optimally manage the loading of objects in memory, allow a locking policy to be set up, etc.

Few environments of this type are sufficiently powerful and complete. The disappearance, at the end of the 90s, of numerous database object editors did not help matters. Meanwhile, the object features integrated by the main editors of relational databases in their products a few years ago are not widely used. Even vendors of O/R mapping tools have experienced difficulties. For example, The Object People had problems before its main business was taken over by WebGain and BEA.

However, we believe that it is a domain that is once again in turmoil (see also [26]).

We can turn to O/R mapping tool such as WebGain's TopLink, Thought's Cocobase or Apple's EOF. We can also turn to object database such as Versant or Objectivity. In this sector, new players are appearing and with them new technical approaches, such as the promising db4o [27]. Some of these products can easily integrate with application servers.

 

## An attempt at standardization

We are currently witnessing an initiative for standardization – the interesting Java Data Object (JDO) specification [28] which, in our opinion, is better suited to object persistence in Java than Entity Beans.

Certain vendors of EJB containers are moving towards using JDOs as an internal machinery so as to ensure the persistence of their Entity Beans. It is also possible to use JDO for the persistence of EJBs in BMP mode instead of JDBC calls. In fact, JDO was designed to support this type of integration with EJB.

But JDO is also a full persistence framework, and it would be a pity to limit it to being no more than a facility for EJB persistence. The JDO model is more powerful, transparent and is lighter than the Entity Bean model suggested in the recent 2.0 draft. Importantly, it is better integrated with the standard Java model. This is why, in most cases, it is more useful to directly use JDO without involving Entity Beans at all (e.g. the JDO API can be directly used from Session Beans, a SOAP listener or a Servlet).

Currently developed as part of the Java Community Process, the JDO specification was designed by experts in the field of object persistence. Supported by small vendors and by SUN, broader support must be found from the industry in order to reach the critical mass.

 

# Conclusion

The arrival of Web services and the willingness to adopt high-level and modern object techniques is shaking up the main standards (COM, CORBA and EJB). Other technologies better suited to the context are emerging. Thanks to them, Web services today provide a technical solution for deployment of distributed applications on the Internet and can be implemented using business object-based architectures.

To conclude this tour of the technological landscape, let us try to look ahead and imagine the Internet when new generation APIs are standard. How will enterprises use Web services to build sales offerings and sign partnerships?

Here, at Orchestra Networks, we believe that a new type of application will appear which will take advantage of Web services. Contrary to traditional applications this new approach will not be directed towards an internal enterprise system but towards an assembly of external multiple systems.

These applications, which we today call Web Business Processes, will be based on the following main characteristics:

## Native connection to Web services

Web services provide a means of standardizing calls between enterprises. Future eBusiness applications will natively implement these standards. In addition, these applications will offer semantic standardization functionalities and stopgap tools for application data mapping.

Certain vendors already offer tools that are highly Web service-oriented such as Bowstreet (Business Web Factory) [29] and Microsoft (.NET).

## Workflow for assembling Web services

The concept of workflow is becoming increasingly important in designing eBusiness applications. Once an enterprise has to assemble services from multiple sources and organize them for the end-user, it must master the process and the sequencing of its system. The asynchronous nature of these applications helps justify the use of workflow engines.

The workflow approach offers numerous services such as execution control, management of application errors, logging and reporting.

Application server vendors are gradually integrating process management components in their tools, such as BEA WebLogic Integration [30].

 

## Separating layers and customization

The closed and monolithic approach will disappear, to the advantage of more open architectures. Web Business Processes clearly separate the application logic, data model, business rules and graphic design of the user interface. Enterprises can thus manipulate and adapt their applications to the varied contexts of their partners and clients.

## Integrating the concept of collaboration

The move from enterprise-oriented applications to those geared towards partnership involves placing the concept of collaboration at the heart of projects. Rules for partnerships and synchronizing business form the basis of Web Business Processes.

**Internet is today entering a phase of technological maturity in which major open standards are adopted. Web services and the underlying technologies are participating in a large-scale movement of cooperation and alliance between enterprises.**

# References

[1] *The Rise of Web Services: Completing the Picture*, Johann Dumser & Jean-Christophe Cimetiere, TrendMarkers, Vol. 3 Number 1, TechMetrix Research. http://www.techmetrix.com/trendmarkers/tmk0101/tmk0101-4.php3

[2] http://www.bea.com/press/releases/2001/0226_web_services.shtml
[3] http://www.alphaworks.ibm.com/tech/wsde
[4] http://www.xmlbus.com/
[5] http://www.themindelectric.com
[6] http://www.shinkatech.com
[7] http://www.idoox.com
[8] http://www.capeclear.com
[9] http://msdn.microsoft.com/net/
[10] http://www-106.ibm.com/developerworks/webservices/
[11] http://www.techmetrix.com/trendmarkers/topics/tmktopic_asp.php3
[12] http://www.webservices.org/
[13] *Bringing Distributed Objects to the World Wide Web*, Ron I. Resnick. http://hegel.ittc.ukans.edu/projects/corba/postscript/javacorb.html
[14] *When is SOAP a good idea in a project*, Billy Newport, TheServerSide.com. http://theserverside.com/resources/article.jsp?l=SOAP-And-EJB
[15] *RMI/IIOP, nice idea but the reality is turning out to be different*, Billy Newport, TheServerSide.com. http://theserverside.com/resources/article.jsp?l=RMI-IIOP
[16] *Application-Level Protocols,* Steve Vinoski, Chief Architect, Iona Technologies. http://www.iona.com/sphere/is0601.html#stevev
[17] *Integrating Apache SOAP with an EJB Server*, Billy Newport, TheServerSide.com. http://theserverside.com/resources/soapandejb1.jsp
[18] *Enterprise JavaBeans Specification version 1.1*, Vlada Matena & Mark Hapner, SUN Microsystems Inc. http://java.sun.com/products/ejb/docs.html
[19] *Designing Entity Beans for Improved Performance*, Beth Stearns. Published by SUN Microsystems Inc. http://developer.java.sun.com/developer/technicalArticles/ebeans/ejbperformance/
[20] *EJBs From a Critical Perspective,* Philippe Mougin, TrendMarkers Volume 1 Number 2, TechMetrix Research, December 1999. http://www.techmetrix.com/trendmarkers/tmk1299/tmk1299-2.php3
[21] Let's Talk EJB, Published by TechMetrix Research, February 2000. http://www.techmetrix.com/trendmarkers/letters/archiveletters/ejb/letejbindex.php3
[22] *JSR-000109 Implementing Enterprise Web Services*, IBM Corporation. http://www.j2ee.com/aboutJava/communityprocess/jsr/jsr_109_implement.html
[23] http://www.soaprpc.com/software/
[24] *Developer's Guide to Building XML-based Web Services with the Java 2 Platform, Enterprise Edition (J2EE)*, James Kao, The Middleware Company. http://developer.java.sun.com/developer/technicalArticles/ebeans/ejbperformance/
[25] http://www.jdom.org
[26] *The Match Game*, Jennifer DeJong, SDTimes. http://sdtimes.com/news/029/special1.htm
[27] http://www.db4o.com/
[28] http://access1.sun.com/jdo/
[29] http://www.bowstreet.com
[30] http://www.bea.com/products/weblogic/integration/index.shtml

 

# About Orchestra Networks

Orchestra Networks provides a web service infrastructure to enable companies to deploy their collaborative commerce strategies.

Orchestra Networks' new generation technological platform enables companies to easily and rapidly form alliances on the Internet and share their expertise to increase their points of presence and create solution packages.

Orchestra Networks' technology and services push back the limits of sales models over the Internet by guaranteeing the publication, integration and driving of collaborative web business processes.

## The Web services revolution

Webservices represent a major development in the e-commerce sector. They enable companies to capitalize on their existing architecture by making their application services accessible via the Internet.

Webservices have been adopted by the principal players in the software industry, and comply with major standards such as XML, SOAP, WSDL and UDDI.

Orchestra Networks offers its customers a c-commerce application development environment based on webservices. This solution enables companies to deploy applications for their partners without redeveloping or outsourcing their key services.

For example, a bank can develop a consumer credit application to be integrated in e-commerce sites. Using the Orchestra Networks environment, the bank will be able to reuse key components of its system, such as simulation or credit scoring modules.

## Driving partnerships

Orchestra Networks enables companies to increase their partnerships over the Internet whilst keeping control over their services. To do this, Orchestra Networks provides operational tools which facilitate:

- Definition of marketing rules for each partner
- Adaptation of service behavior

These operational tools are for use by marketing teams and do not necessarily require advanced computer skills.

 

# Seamless integration

Orchestra Networks enables enterprises to seamlessly integrate their offering in partner web sites. Orchestra Networks supplies partner web sites with customization and merchandising tools which enable:

- Complete respect of their graphic guidelines
- Dynamic configuration of the service according to the customer context
- Optimization of positioning of the offering on their web site

The customization tools are used via the Internet and do not require any development on the partner site's server. On average, integration of a service in a web site requires less than half a day of configuration time.

Further information may be found on:

www.**orchestranetworks**.com

 

# About SQLI -TechMetrix Research

TechMetrix Research is a technically-oriented analyst firm focused on e-business application development needs.
TechMetrix Research has developed a unique evaluation approach to provide accurate information on software.
With location in the US (Boston, MA), France (Paris, Lyon) and Switzerland (Lausanne), the firm publishes comparison reports, product reviews and technical analysis, which are real helpers when it comes to making decisions, or simply keeping pace with the fast moving e-business market.

As its parent company (SQLI) provides information system development and implementation services to major companies, TechMetrix Research also benefits from the feedback and experience acquired during large-scale, long-term development projects.

Groupe SQLI is a European global system integrator of 700 employees offering full service and continuing coaching to enable companies to move profitably toward an all-Internet solution.
Groupe SQLI helps clients to define and implement a successful Internet strategy and offer all the skills needed: Web marketing, design, usability, the three-pronged philosophy (reliability, fluidity and ease) of KeenVision (www.keenvision.com), technology consulting, workload tests by TechMetrix Research (www.techmetrix.com) and development of optimal and efficient IT solutions by SQLI (www.sqli.com).

**For more information check our websites:**

www.sqli.com www.techmetrix.com www.keenvision.com

 

A White Paper written by

# Orchestra Networks

**Authors**

Philippe Mougin, software architect at Orchestra Networks
philippe.mougin@orchestranetworks.com
Christophe Barriolade,CEO of Orchestra Networks
christophe.barriolade@orchestranetworks.com

**Contributor**

Jean-Christophe Cimetiere, CEO of TechMetrix Research
jcc@techmetrix.com

**Foreword by**

Alain Lefebvre
Jean-Christophe Cimetiere

**Translation**

Hannah Riley
Brid Marire

Publication date: July 2001

Orchestra Networks is the vendor of a technology for collaborative commerce based on Web services. Based in France, Orchestra Networks is building a new generation software infrastructure for developing and deploying partnerships between enterprises on the Internet.

www.**orchestranetworks**.com

 